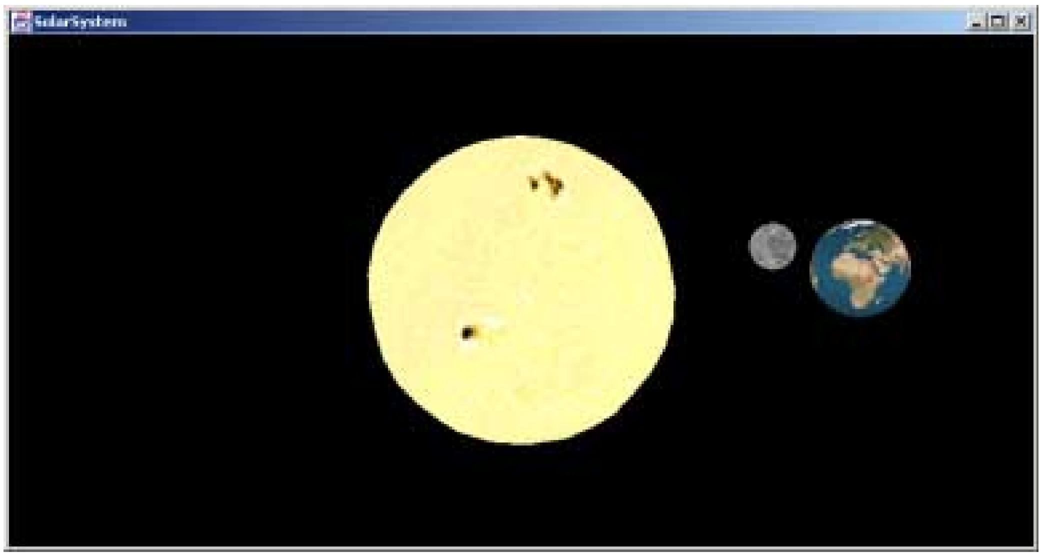
Elementary Java3D

During this lab exercise, you will be introduced to the Java3D graphics API. Java3D is a high-level, scene graph-based framework for 3D graphics. Your task is to write a program that displays an animated planetary system with a sun in the middle, a spinning planet in a circular orbit around the sun, and a spinning moon in a circular orbit around the planet. The amount of code you will need to write is quite small, but the amount of thought that needs to be put into it is considerable. The task is selected to give a reasonably interesting but simple display, and to clarify the concepts of the Java3D scene graph, hierarchical transformations, animation behaviors and object appearance without going into too much detail.



Preparation

Doing anything at all with Java3D requires you to be reasonably familiar with the Java3D scene graph concept, so **start your preparations by reading the first chapter of the Java3D tutorial**, available at the following address:

http://java.sun.com/products/java-media/3D/collateral/j3d_tutorial_ch1.pdf

For your convenience, we have also placed a local copy on:

http://staffwww.itn.liu.se/~stegu/TNM061-2007/java3d/

The chapter is about 30 pages, but please read all of it. It is a brief and good introduction to Java3D, and you will need to have a reasonable grip on its contents to complete this lab exercise successfully.

If you encounter any problems during the lab session, you are most welcome to ask for help, but the intention is for you to solve this on your own, in your own way, so don't expect more than general hints from the lab assistant. We also reserve the right to refuse to answer questions that are obviously due to inappropriate preparations, for example not having read chapter 1 of the Java3D tutorial at all.

If you encounter any *real* problems, we are certainly always open for questions, but the small obstacles on the way to success are there for you to solve them all by yourself. Have fun, and always remember to do two things: *read* and *think*. It helps a lot.

Bug reports, suggestions for improvement and general comments on this exercise are much appreciated. Good luck!

Stefan Gustavson (stegu@itn.liu.se)

Lab assignments

The final result of this exercise, i.e. what we want you to show in order to pass the assignment, is an animated view of a planetary system with a yellow sun at the origin, a textured planet rotating around it, and a textured moon rotating around the planet, resembling the image on the previous page.

In order to make the task more manageable, we suggest you split it up into the following incremental steps. After you you have completed step 8, you have performed the required part of the exercise. Steps 9 to 12 are some suggested extra assignments which can be performed in any order to explore some of the further possibilities of Java3D.

1. Run and examine the source code for the class `HelloUniverse.java` from Sun. You can find it on: http://staffwww.itn.liu.se/~stegu/TNM061-2007/java3d/
That class gives you all the basic functionality for setting up a 3D window and animating a simple object in it. Make sure you thoroughly understand all of the scene graph stuff in the method `createScene-Graph()`, and have a brief look at the other methods to see what they do.
What is the scene graph for this simple example? **Draw it!**

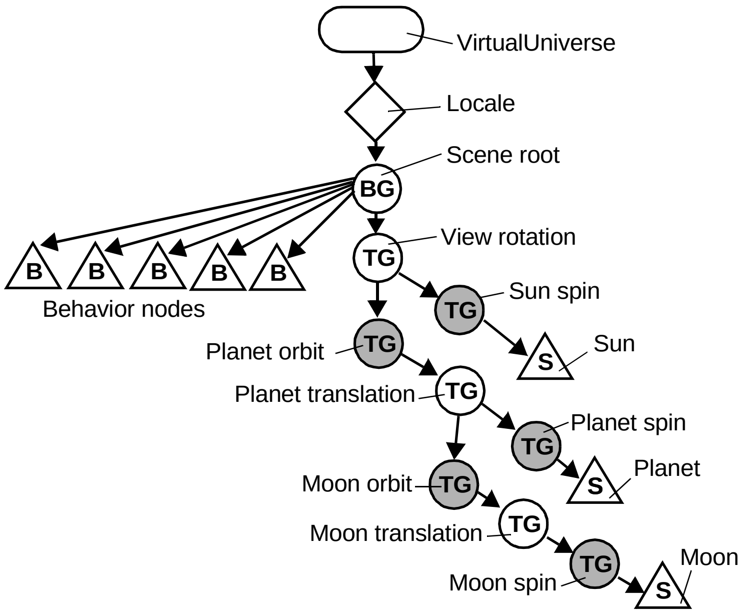
2. Modify the animation to offset the object from the origin, and rotate it in an orbit around the origin. As you surely remember, this can be done by a hierarchical transformation: a static translation along the X axis followed by an animated rotation around the Y axis.

3. Add a small static rotation around the X axis at the root of the scene graph to view the orbiting cube from slightly above the rotational plane. This makes the image easier to interpret and more interesting.

4. Now, make the cube orbit around the origin *and* rotate around its axis, with different speeds. This requires you to set up two `Alpha` objects to control two `RotationInterpolator` objects. What is the scene graph for your modified program? **Draw it!**

5. Create a larger scene graph tree with a number of `transformGroup` nodes in a hierarchy as shown in the figure below. Set all the animated rotations to different speeds. For the time being, use `ColorCube` objects of different sizes for the sun, the planet and the moon. Start by doing just the sun and the planet, and add the moon branch after you have the sun and planet working.

Scene graph for a simple planetary system. The five shaded transform group nodes (TG) are animated and should each be controlled by the five behavior nodes, each a `RotationInterpolator` object. The implicit connections between the behavior nodes and their corresponding transform groups are not shown in this graph.



6. Replace the rotating cubes with spheres. There is a predefined `Sphere` object in Java3D for you to use. A simple sphere is created in much the same way as a ColorCube:

```
Sphere mysphere = new Sphere(0.5f);
```

There are a large number of additional constructors for `Sphere` to control its detailed appearance.

7. Put surface textures on the planet and the moon to make them look more interesting, and to make the spinning animation visible again. Textures are introduced in chapter 7 of the Java3D tutorial, but a simple example of how to put a texture on a sphere is provided in the file `HelloEarth.java`, available at: http://staffwww.itn.liu.se/~stegu/TNM061-2007/java3d/
Texture images of size 512x256 pixels for the Sun, Earth and Moon can be found on the same address. The images were prepared from the excellent high resolution maps made by James Hastings-Trew, available for free from his site at http://apollo.spaceports.com/~jhasting/.

8. Set values for the rotation speeds to mimic the Sun, Earth, and Moon. Make the Earth spin once every second and rotate one revolution around the Sun in 365 seconds, make the Sun spin a full turn in 25 seconds, and make the moon rotate around the Earth in 27.3 seconds while always showing the same side towards the Earth. Do not try to match the size of the spheres and their relative distances to the corresponding real values. Place them close together and make them only mildly different in size, or else you will get a *very* boring image. If you are interested, real data for the Sun, Earth and Moon is given below.

9. *(Extra)* Make the system look even more like the Earth and Moon by tilting the planet's axis of rotation 23.44 degrees. Make the orientation of the Earth's axis of rotation stay fixed relative to the universe, and do *not* tilt the orbit of the Moon. (The Moon's orbit actually tilts 5.15 degrees around a different axis, but let's ignore that.) This requires the scene branch of the Earth to be restructured somewhat, but the entire branch from the scene root to the Moon should still be kept intact. You will need some additional, carefully designed transform groups, but you shouldn't need any new Behavior nodes.

10. *(Extra)* Place a light source at the position of the sun and make the planet and the moon receive light from it. To find out how to do this, read chapter 6 of the Java3D tutorial, and/or have a look at the appropriate Java3D examples from Sun. Lights are included as leafs in the scene graph, like other object nodes, but in order for them to have any effect on other objects, those objects must be explicitly set up to take light sources into consideration.

11. *(Extra)* If you want, explore the use of mipmapping and texture filtering to get rid of the aliasing for the textures, particularly noticeable on the moon texture. Mipmapping and texture filtering is described in chapter 7 of the Java3D tutorial.

12. *(Extra)* Transform the camera (attach the view platform to a transform group) so that the view is not from outer space, but from a point slightly above the Earth's equator, staying fixed relative to the Earth's surface and looking east at the sky. Sit back and watch the Sun rise every morning. If you have turned on lighting, you can watch days and nights pass on the surface, and even observe the phases of the moon. Depending on how you placed the Moon in your orbit, you may need to rotate it to make it face the correct way, with the familiar large dark features, the "seas", towards the Earth. For this assignment, you may also want to adjust the proportions of your solar system to match the astronomical data below.

(More data would be required for an accurate simulation of the rather complicated planetary movements, but this table lists the most prominent effects. Can you figure out why Earth's rotation time is not 1 day?)

	Sun	Earth	Moon
radius	1 392 000 km	6 378 km	1 738 km
rotation time	25.38 days at equator	0.99726 days	27.3 days
orbit radius	-	149 600 000 km	384 400 km
orbit time	-	365.256 days	27.3 days
axis tilt against orbit	-	23.44 degrees	~ 0
orbit tilt against Earth's orbit	-	0 (by definition)	5.15 degrees